-- --

IEEE P1003.2 Draft 11.2 – September 1991

(2) All of the calls shall set GLOB_DOOFFS, or all shall not set it. 1

(3) After the second call, *pglob->gl_pathv* shall point to a list containing the following:

(a) Zero or more **NULL** s, as specified by GLOB_DOOFFS and *pglob->gl_offs*.

(b) Pointers to the pathnames that were in the *pglob->gl_pathv* list before the call, in the same order as before.

(c) Pointers to the new pathnames generated by the second call, in the specified order.

(4) The count returned in *pglob->gl_pathc* shall be the total number of pathnames from the two calls.

The application can change any of the fields in Table B-12 after a call to *glob*(), 1
but if it does it shall reset them to the original value before a subsequent call, 1
using the same *pglob* value, to *globfree*() or *glob*() with the GLOB_APPEND flag. 1

If, during the search, a directory is encountered that cannot be opened or read and *errfunc* is not **NULL**, *glob*() shall call (**errfunc*)() with two arguments:

(1) The *epath* argument is a pointer to the path that failed.

(2) The *eerrno* argument is the value of *errno* from the failure, as set by the POSIX.1 {8} *opendir*(), *readdir*(), or *stat*() functions. (Other values may be used to report other errors not explicitly documented for those functions.)

If (**errfunc*)() is called and returns nonzero, or if the GLOB_ERR flag is set in *flags*, *glob*() shall stop the scan and return GLOB_ABORTED after setting *gl_pathc* and *gl_pathv* in *pglob* to reflect the paths already scanned. If GLOB_ERR is not set and either *errfunc* is **NULL** or (**errfunc*)() returns zero, the error shall be ignored.

B.8.3 Returns

On successful completion, *glob*() shall return zero. The argument *pglob->gl_pathc* shall return the number of matched pathnames and the argument *pglob->gl_pathv* shall contain a pointer to a null-terminated list of matched and sorted pathnames. However, if *pglob->gl_pathc* is zero, the content of *pglob->gl_pathv* is undefined.

B.8.4 Errors

If *glob*() terminates due to an error, it shall return one of the nonzero constants shown in Table B-14, which are defined in <glob.h>. The arguments *pglob->gl_pathc* and *pglob->gl_pathv* are still set as defined above in Returns.

Table B-14 – *glob* () Error Return Values

Name	Description	
GLOB_ABORTED	The scan was stopped because GLOB_ERR was set or (*errfunc)() returned nonzero.	
GLOB_NOMATCH	The *pattern* does not match any exiting pathname, and GLOB_NOCHECK was not set in *flags*.	1 1
GLOB_NOSPACE	An attempt to allocate memory failed.	

B.8.5 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

This function is not provided for the purpose of enabling utilities to perform pathname expansion on their arguments, as this operation is performed by the shell, and utilities are explicitly not expected to redo this. Instead, it is provided for applications that need to do pathname expansion on strings obtained from other sources, such as a pattern typed by a user or read from a file.

If a utility needs to see if a pathname matches a given pattern, it can use *fnmatch* ().

Note that *gl_pathc* and *gl_pathv* have meaning even if *glob*() fails. This allows *glob*() to report partial results in the event of an error. However, if *gl_pathc* is zero, *gl_pathv* is unspecified even if *glob*() did not return an error.

The GLOB_NOCHECK option could be used when an application wants to expand a pathname if wildcards are specified, but wants to treat the pattern as just a string otherwise. The `sh` utility might use this for option-arguments, for example.

One use of the GLOB_DOOFFS flag is by applications that build an argument list for use with the POSIX.1 {8} *execv*(), *execve*(), or *execvp*() functions. Suppose, for example, that an application wants to do the equivalent of `ls -l *.c`, but for some reason `system("ls -l *.c")` is not acceptable. The application could obtain (*approximately*) the same result using the sequence:

```
globbuf.gl_offs = 2;
glob ("*.c", GLOB_DOOFFS, NULL, &globbuf);
globbuf.gl_pathv[0] = "ls";
globbuf.gl_pathv[1] = "-l";
execvp ("ls", &globbuf.gl_pathv[0]);
```

Using the same example, `ls -l *.c *.h` could be approximately simulated using GLOB_APPEND as follows:

```
globbuf.gl_offs = 2;
glob ("*.c", GLOB_DOOFFS, NULL, &globbuf);
glob ("*.h", GLOB_DOOFFS|GLOB_APPEND, NULL, &globbuf);
 ... etc. ...
```

The new pathnames generated by a subsequent call with GLOB_APPEND are not sorted together with the previous pathnames. This mirrors the way that the shell handles pathname expansion when multiple expansions are done on a command line.

History of Decisions Made

The interface was simplified to a useful, but less complex, subset. The *errfunc* argument was added to allow errors to be reported.

A reviewer claimed that the GLOB_DOOFFS flag is unnecessary because it could be simulated using:

```
new = (char **)malloc((n + pglob->gl_pathc + 1)
                * sizeof (char *));
(void) memcpy (new+n, pglob->gl_pathv,
                pglob->gl_pathc * sizeof(char *));
(void) memset (new, 0, n * sizeof (char *));
free (pglob->gl_pathv);
pglob->gl_pathv = new;
```

However, this assumes that the memory pointed to by *gl_pathv* is a block that was separately created using *malloc* (). This is not necessarily the case. An application should make no assumptions about how the memory referenced by fields in *pglob* was allocated. It might have been obtained from *malloc* () in a large chunk, and then carved up within *glob* (), or it might have been created using a different memory allocator. It is not the intent of this standard to specify or imply how the memory used by *glob* () is managed.

The structure elements *gl_pathc* and *gl_pathv* were renamed from *gl_argc* and *gl_argv* in Draft 9. The old names implied an association with the parameters to *main* () that does not necessarily exist.

The GLOB_APPEND flag was added in Draft 9 at the request of a reviewer. This flag would be used when an application wants to expand several different patterns into a single list.

Tilde and parameter expansion were removed from *glob* () in Draft 9. Applications that need these expansions should use the *wordexp* () function [see B.9].

B.9 C Binding for Perform Word Expansions

Functions: *wordexp* (), *wordfree* ()

B.9.1 Synopsis

```
#include <wordexp.h>
int wordexp(const char *words, wordexp_t *pwordexp, int flags);
void wordfree(wordexp_t *pwordexp);
```

B.9.2 Description

The *wordexp* () function shall perform word expansions as described in 3.6, subject to quoting as in 3.2, and place the list of expanded words into *pwordexp* . The expansions shall be the same as would be performed by the shell if *words* were the part of a command line representing the arguments to a utility. Therefore, *words* shall not contain an unquoted <newline> or any of the unquoted shell special characters |, &, ;, <, or >, except in the context of command substitution as specified in 3.6.3. It also shall not contain unquoted parentheses or braces, except in the context of command or variable substitution. If *words* contains an unquoted comment character (number sign) that is the beginning of a token, *wordexp* () may treat the comment character as a regular character, or may interpret it as a comment indicator and ignore the remainder of *words* .

The header <wordexp.h> defines the structure type *wordexp_t* , which includes at least the members shown in Table B-15.

Table B-15 – Structure Type *wordexp_t*

Member Type	Member Name	Description	
size_t	*we_wordc*	Count of words matched by *words* .	1
char **	*we_wordv*	Pointer to list of expanded words.	
size_t	*we_offs*	Slots to reserve at the beginning of *we_wordv* .	1

The argument *words* is a pointer to a string containing one or more words to be expanded. The *wordexp* () function shall store the number of generated words into *we_wordc* and a pointer to a list of pointers to words in *we_wordv* . Each individual field created during field splitting (see 3.6.5) or pathname expansion (see 3.6.6) is a separate word in the *we_wordv* list. The words are in order as described in 3.6. The first pointer after the last word pointer shall be **NULL** . The expansion of special parameters described in 3.5.2 is unspecified.

It is the caller's responsibility to create the structure pointed to by *pwordexp* . The *wordexp* () function allocates other space as needed, including memory pointed to by *we_wordv*. The *wordfree* () function shall free any memory associated with *pwordexp* from a previous call to *wordexp* ().

The argument *flags* is used to control the behavior of *wordexp* (). The value of *flags* is the bitwise inclusive OR of any of the constants in Table B-16, which are defined in `<wordexp.h>`.

Table B-16 – *wordexp* () *flags* Argument

Name	Description
WRDE_APPEND	Append words generated to the ones from a previous call to *wordexp* ().
WRDE_DOOFFS	Make use of *we_offs*. If this flag is set, *we_offs* is used to specify how many **NULL** pointers to add to the beginning of *we_wordv*. In other words, *we_wordv* shall point to *we_offs* **NULL** pointers, followed by *we_wordc* word pointers, followed by a **NULL** pointer.
WRDE_NOCMD	Fail if command substitution, as specified in 3.6.3, is requested.
WRDE_REUSE	The *pwordexp* argument was passed to a previous successful call to *wordexp* (), and has not been passed to *wordfree* (). The result shall be the same as if the application had called *wordfree* () and then called *wordexp* () without WRDE_REUSE.
WRDE_SHOWERR	Do not redirect standard error to `/dev/null`.
WRDE_UNDEF	Report error on an attempt to expand an undefined shell variable.

The WRDE_APPEND flag can be used to append a new set of words to those generated by a previous call to *wordexp* (). The following rules apply when two or more calls to *wordexp* () are made with the same value of *pwordexp* and without intervening calls to *wordfree* ():

(1) The first such call shall not set WRDE_APPEND. All subsequent calls shall set it.

(2) All of the calls shall set WRDE_DOOFFS, or all shall not set it.

(3) After the second and each subsequent call, *we_wordv* shall point to a list containing the following:

  (a) Zero or more **NULL** s, as specified by WRDE_DOOFFS and *we_offs*.

  (b) Pointers to the words that were in the *we_wordv* list before the call, in the same order as before.

  (c) Pointers to the new words generated by the latest call, in the specified order.

(4) The count returned in *we_wordc* shall be the total number of words from all of the calls.

The application can change any of the fields in Table B-15 after a call to *wor-* 1
dexp (), but if it does it shall reset them to the original value before a subsequent 1
call, using the same *pwordexp* value, to *wordfree* () or *wordexp* () with the 1
WRDE_APPEND or WRDE_REUSE flag. 1

If *words* contains an unquoted `<newline>`, |, &, `;`, <, >, parenthesis, or brace in an inappropriate context, *wordexp* () shall fail, and the number of expanded words shall be zero.

Unless WRDE_SHOWERR is set in *flags*, *wordexp* () shall redirect standard error to `/dev/null` for any utilities executed as a result of command substitution while expanding *words*. If WRDE_SHOWERR is set, *wordexp* () may write messages to standard error if syntax errors are detected while expanding *words*.

If WRDE_DOOFFS is set, then *we_offs* shall have the same value for each *wor-* 1
dexp () call and the *wordfree* () call using a given *pglob*. 1

B.9.3 Returns

If no errors are encountered while expanding *words*, *wordexp* () shall return zero. Otherwise it shall return a nonzero value.

B.9.4 Errors

Table B-17 – *wordexp* () Return Values

Name	Description
WRDE_BADCHAR	One of the unquoted characters |, &, `;`, <, >, parentheses, or braces appears in *words* in an inappropriate context.
WRDE_BADVAL	Reference to undefined shell variable when WRDE_UNDEF is set in *flags*.
WRDE_CMDSUB	Command substitution requested when WRDE_NOCMD was set in flags.
WRDE_NOSPACE	Attempt to allocate memory failed
WRDE_SYNTAX	Shell syntax error, such as unbalanced parentheses or unterminated string.

If *wordexp* () terminates due to an error, it shall return one of the nonzero constants shown in Table B-17, which shall be defined in `<wordexp.h>`. The implementation may define additional error returns beginning with WRDE_.

If *wordexp* () returns the error value WRDE_NOSPACE, then *pwordexp->we_wordc* and *pwordexp->we_wordv* shall be updated to reflect any words that were successfully expanded. In other cases, they shall not be modified.

B.9.5 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

This function is intended to be used by an application that wants to do all of the shell's expansions on a word or words obtained from a user. For example, if the application prompts for a file name (or list of file names) and then used *wordexp* () to process the input, the user could respond with anything that would be valid as input to the shell.

The WRDE_NOCMD flag is provided for applications that, for security or other reasons, want to prevent a user from executing shell commands. Disallowing unquoted shell special characters also prevents unwanted side effects such as executing a command or writing a file.

History of Decisions Made

This function was added in Draft 9 as an alternative to *glob* (). There has been continuing controversy over exactly what features should be included in *glob* (). It is hoped that providing *wordexp* () (which provides all of the shell's word expansions, but will probably be slow to execute), and *glob* () (which is faster but does only expansion of pathnames, without tilde or parameter expansion), will satisfy the majority of reviewers.

While *wordexp* () could be implemented entirely as a library routine, it is expected 1
that most implementations will run a shell in a subprocess to do the expansion.

Two different approaches have been proposed for how the required information might be presented to the shell and the results returned. They are presented here as examples.

One proposal is to extend the `echo` utility by adding a `-q` option. This option would cause `echo` to add a backslash before each backslash and each `<blank>` that occurs within an argument. The *wordexp* () function could then invoke the shell as follows:

```
(void) strcpy (buffer, "echo -q ");
(void) strcat (buffer, words );
if ((flags & WRDE_SHOWERR) == 0)
        (void) strcat (buffer, " 2>/dev/null");
f = popen (buffer, "r");
```

The *wordexp* () function would read the resulting output, remove unquoted backslashes, and break into words at unquoted `<blank>`s. If the WRDE_NOCMD flag was set, *wordexp* () would have to scan *words* before starting the subshell to make sure that there would be no command substitution. In any case, it would have to scan *words* for unquoted special characters.

Another proposal is to add the following options to `sh`:

`−w` *wordlist* This option provides a wordlist expansion service to applications. The words in *wordlist* are expanded, and the following is written to standard output:

(1) The count of the number of words after expansion, in decimal, followed by a null byte.

(2) The number of bytes needed to represent the expanded words (not including null separators), in decimal, followed by a null byte.

(3) The expanded words, each terminated by a null byte.

If an error is encountered during word expansion, `sh` exits with a nonzero status after writing the above to report any words successfully expanded

`−P` Run in ‘‘protected’’ mode. If specified with the `−w` option, no command substitution is performed.

With these options, *wordexp* () could be implemented fairly simply by creating a subprocess using *fork* (), and executing `sh` using the line:

```
execl(<shell path>, "sh", "-P", "-w", words, (char *)0);
```

after directing standard error to `/dev/null`.

It seemed objectionable for a library routine to write messages to standard error, unless explicitly requested, so *wordexp* () is required to redirect standard error to `/dev/null` to ensure that no messages are generated, even for commands executed for command substitution. The new WRDE_SHOWERR flag can be specified to request that error messages be written.

The WRDE_REUSE flag allows the implementation to avoid the expense of freeing and reallocating memory, if that is possible. A minimal implementation can just call *wordfree* () when WRDE_REUSE is set.

B.10 C Binding for Get POSIX Configurable Variables

B.10.1 C Binding for Get String-Valued Configurable Variables

Function: *confstr* ()

B.10.1.1 Synopsis

```
#include <unistd.h>
size_t confstr(int name, char *buf, size_t len);
```

B.10.1.2 Description

The *confstr* () function provides a method for applications to get configuration-defined string values. Its use and purpose are similar to the *sysconf* () function defined in POSIX.1 {8}, but it is used where string values rather than numeric values are returned.

The *name* argument represents the system variable to be queried. The implementation shall support all of the *name* values shown in Table B-18, which are defined in <unistd.h>. It may support others.

Table B-18 – confstr() *name* Values

name **Value**	**String returned by confstr()**
_CS_PATH	A value for the **PATH** environment variable that finds all standard utilities.

If *len* is not zero, and if *name* has a configuration-defined value, *confstr* () shall copy that value into the *len*-byte buffer pointed to by *buf*. If the string to be returned is longer than *len* bytes, including the terminating null, then *confstr* () shall truncate the string to *len*−1 bytes and null-terminate the result. The application can detect that the string was truncated by comparing the value returned by *confstr* () with *len*.

If *len* is zero and *buf* is **NULL**, then *confstr* () still shall return the integer value as defined below, but shall not return a string. If *len* is zero but *buf* is not **NULL**, the result is unspecified.

B.10.1.3 Returns

If *name* does not have a configuration-defined value, *confstr* () shall return zero and leave *errno* unchanged.

If *name* has a configuration-defined value, the *confstr* () function shall return the size of buffer that would be needed to hold the entire configuration-defined value.

If this return value is greater than *len*, the string returned in *buf* has been truncated.

B.10.1.4 Errors

If any of the following conditions occur, *confstr* () shall return zero and set *errno* to the corresponding value:

[EINVAL] The value of the *name* argument is invalid.

B.10.1.5 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

An application can distinguish between an invalid *name* parameter value and one that corresponds to a configurable variable that has no configuration-defined value by checking if *errno* has been modified. This mirrors the behavior of *sysconf* () in POSIX.1 {8}.

The original need for this function was to provide a way of finding the configuration-defined default value for the environment variable **PATH**. Since **PATH** can be modified by the user to include directories that could contain utilities replacing POSIX.2 standard utilities, applications need a way to determine the system-supplied **PATH** environment variable value that contains the correct search path for the POSIX.2 standard utilities.

An application could use `confstr(name,NULL,(size_t) 0)` to find out how big a buffer is needed for the string value, *malloc* () a buffer to hold the string, and call *confstr* () again to get the string. Alternately, it could allocate a fixed, static buffer that is big enough to hold most answers (512 bytes, maybe, or 1024), but then *malloc* () a larger buffer if it finds that this is too small.

History of Decisions Made

In Draft 7, these values and *sysconf* () values defined in POSIX.1 {8} were obtained using a function named *posixconf* (). However, that routine was dropped in favor of *csysconf* (). There did not seem to be any reason to provide the redundant interface to POSIX.1 {8} functions, nor to return values as strings when numeric values are really what are needed. *csysconf* () could be extended to return strings for other related standards or features.

In Draft 9, *csysconf* () has been replaced by *confstr* (). The name was changed because too many people were confused by the name; they thought that the 'c' referred to the C language, rather than characters (as distinct from integers). The *confstr* () function also copies the returned string into a buffer supplied by the application instead of returning a pointer to a string. This allows a cleaner interface in some implementations (lightweight processes were mentioned), and resolves questions about when the application must copy the string returned.

B.10.2 C Binding for Get Numeric-Valued Configurable Variables

Functions: *sysconf* (), *pathconf* (), *fpathconf* ()

A system that supports the C Language Bindings Option shall support the C language bindings defined in POSIX.1 {8} for the *sysconf* (), *pathconf* (), and *fpathconf* () functions. Of the *name* values defined in POSIX.1 {8}, only those that correspond to numeric-valued configuration values listed in Table 7-1, are required by POSIX.2. In addition, the *sysconf* () function shall support the *name* values in Table B-19, defined in `<unistd.h>`, to provide values for values in 2.13.1.

Table B-19 – C Bindings for Numeric-Valued Configurable Variables

Symbolic Limit	*name* Value
{BC_BASE_MAX}	_SC_BC_BASE_MAX
{BC_DIM_MAX}	_SC_BC_DIM_MAX
{BC_SCALE_MAX}	_SC_BC_SCALE_MAX
{BC_STRING_MAX}	_SC_BC_STRING_MAX
{COLL_WEIGHTS_MAX}	_SC_COLL_WEIGHTS_MAX
{EXPR_NEST_MAX}	_SC_EXPR_NEST_MAX
{LINE_MAX}	_SC_LINE_MAX
{RE_DUP_MAX}	_SC_RE_DUP_MAX
{POSIX2_VERSION}	_SC_2_VERSION
{POSIX2_C_DEV}	_SC_2_C_DEV
{POSIX2_FORT_DEV}	_SC_2_FORT_DEV
{POSIX2_FORT_RUN}	_SC_2_FORT_RUN
{POSIX2_LOCALEDEF}	_SC_2_LOCALEDEF
{POSIX2_SW_DEV}	_SC_2_SW_DEV

B.10.3 Rationale. *(This subclause is not a part of P1003.2)*

In Draft 9, the *name* values corresponding to the _POSIX2_* symbolic limits were changed to more closely follow the convention used in POSIX.1 {8}. In POSIX.1 {8}, for example, the *name* value for {_POSIX_VERSION} is _SC_VERSION. The POSIX.2 *name* value for {_POSIX2_C_DEV} (actually, it was {_POSIX_C_DEV} in Draft 8) was _SC_POSIX_C_DEV, and is now _SC_2_C_DEV.

If `sysconf(_SC_2_VERSION)` is not equal to the value of the {_POSIX2_VERSION} symbolic constant (see B.2.2), the utilities available via *system* () or *popen* () might not behave as described in this standard. This would mean that the application is not running in an environment that conforms to POSIX.2. Some applications might be able to deal with this, others might not. However, the interfaces defined in Annex B shall continue to operate as specified, even if `sysconf(_SC_2_VERSION)` reports that the utilities no longer perform as specified.

B.11 C Binding for Locale Control

The C binding to the services described in 7.9 shall be the *setlocale* () function defined in POSIX.1 {8} 8.1.2. In addition to the category values defined in POSIX.1 {8}, *setlocale* () shall also accept the value LC_MESSAGES, which shall be defined in `<locale.h>`.

B.11.1 C Binding for Locale Control Rationale. *(This subclause is not a part of P1003.2)*

The order in which the various locale categories are processed by *setlocale* () is not specified by POSIX.1 {8}, so the place for LC_MESSAGES in that order is also unspecified.

Annex C
(normative)

FORTRAN Development and Runtime Utilities Options

This annex describes utilities used for the development of FORTRAN language applications, including compilation or translation of FORTRAN source code, and the execution of certain FORTRAN applications at runtime.

The utilities described in this annex may be provided by the conforming system; however, any system claiming conformance to the FORTRAN Development Utilities Option shall provide the `fort77` utility and any system claiming conformance to the FORTRAN Runtime Utilities Option shall provide the `asa` utility.

C.0.1 FORTRAN Development and Runtime Utilities Options Rationale.

(This subclause is not a part of P1003.2)

This clause is included in this standard as a temporary measure to accommodate existing FORTRAN developers. It is the intention of the POSIX.2 working group that this annex be moved from this standard to the emerging standard being developed by the POSIX.9 working group, which will specify FORTRAN-specific interfaces to the basic services provided by this standard and POSIX.1. The movement of this annex should occur in a later version of this standard.

See the rationale for `asa` for a description of the FORTRAN Runtime Utilities Option and why it was split off from the FORTRAN Development Utilities Option.

C.1 `asa` — Interpret carriage-control characters

This utility is optional. It shall be provided on systems that support the FORTRAN Runtime Utilities Option.

C.1.1 Synopsis

`asa` [*file* ...]

C.1.2 Description

The `asa` utility shall write its input files to standard output, mapping carriage-control characters from the text files to line-printer control sequences in an implementation-defined manner.

The first character of every line shall be removed from the input, and the following actions shall be performed:

If the character removed is:

`<space>` The rest of the line shall be output without change.

`0` A `<newline>` shall be output, then the rest of the input line.

`1` One or more implementation-defined characters that causes an advance to the next page shall be output, followed by the rest of the input line.

`+` The `<newline>` of the previous line shall be replaced with one or more implementation-defined characters that causes printing to return to column position 1, followed by the rest of the input line. If the `+` is the first character in the input, it shall have the same effect as `<space>`.

The action of the `asa` utility is unspecified upon encountering any character other than those listed above as the first character in a line.

C.1.3 Options

None.

C.1.4 Operands

file A pathname of a text file used for input. If no *file* operands are specified, the standard input shall be used.

C.1.5 External Influences

C.1.5.1 Standard Input

The standard input shall be used only if no *file* operands are specified. See Input Files.

C.1.5.2 Input Files

The input files shall be text files.

C.1.5.3 Environment Variables

The following environment variables shall affect the execution of `asa`:

LANG This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).

LC_MESSAGES This variable shall determine the language in which messages should be written.

C.1.5.4 Asynchronous Events

Default.

C.1.6 External Effects

C.1.6.1 Standard Output

The standard output shall be the text from the input file modified as described in C.1.2.

C.1.6.2 Standard Error

None.

C.1.6.3 Output Files

None.

C.1.7 Extended Description

None.

C.1.8 Exit Status

The `asa` utility shall exit with one of the following values:

- 0 All input files were output successfully.
- >0 An error occurred.

C.1.9 Consequences of Errors

Default.

C.1.10 Rationale. *(This subclause is not a part of P1003.2)*

Examples, Usage

The `asa` utility is needed to map ''standard'' FORTRAN 77 output into a form acceptable to contemporary printers. Usually `asa` is used to pipe data to the `lp` utility (see `lp` in 4.38.)

The following command:

```
asa file
```

permits the viewing of `file` (created by a program using FORTRAN -style carriage control characters) on a terminal.

The following command:

```
a.out | asa | lp
```

formats the FORTRAN output of `a.out` and directs it to the printer.

History of Decisions Made

This utility is generally used only by FORTRAN programs. It was moved to this annex in response to multiple ballot objections requesting its removal. The working group decided to retain `asa` to avoid breaking the existing large base of FORTRAN applications that put carriage control characters in their output files. This is a compromise position to achieve balloting acceptance: the overhead of maintaining a separate option in POSIX. 2 for just this one utility is seen to be small in comparison to the benefit achieved for FORTRAN applications. Since it is a separate option, there is no requirement that a system have a FORTRAN compiler in order to run applications that need `asa`.

Historical implementations have used an ASCII `<form-feed>` character in response to a `'1'`, and an ASCII `<carriage-return>` in response to a `'+'`. It is suggested that implementations treat characters other than `'0'`, `'1'`, and `'+'` as `<space>` in the absence of any compelling reason to do otherwise. However, the action is listed here as ''unspecified,'' permitting an implementation to provide extensions to access fast multiple line slewing and channel seeking in a nonportable manner.

C.2 `fort77` — FORTRAN compiler

This utility is optional. It shall be provided on systems that support the FORTRAN Development Utilities Option.

C.2.1 Synopsis

`fort77` [`–c`] [`–g`] [`–L` *directory*] . . . [`–O` *optlevel*] [`–o` *outfile*] [`–s`] [`–w`]
operand . . .

C.2.2 Description

The `fort77` utility is the interface to the FORTRAN compilation system; it shall accept the full FORTRAN language defined by ISO 1539 {2}. The system conceptually consists of a compiler and link editor. The files referenced by *operand* s are compiled and linked to produce an executable file. (It is unspecified whether the linking occurs entirely within the operation of `fort77`; some systems may produce objects that are not fully resolved until the file is executed.)

If the `–c` option is present, for all pathname operands of the form *file* `.f`, the files

`$(basename` *pathname* `.f).o`

shall be created or overwritten as the result of successful compilation. If the `–c` option is not specified, it is unspecified whether such `.o` files are created or deleted for the *file* `.f` operands.

If there are no options that prevent link editing (such as `–c`) and all operands compile and link without error, the resulting executable file shall be written into the file named by the `–o` option (if present) or to the file `a.out`. The executable file shall be created as specified in 2.9.1.4, except that the file permissions shall be set to

S_IRWXO | S_IRWXG | S_IRWXU

(see POSIX.1 {8} 5.6.1.2) and that the bits specified by the *umask* of the process shall be cleared.

C.2.3 Options

The `fort77` utility shall conform to the utility argument syntax guidelines described in 2.10.2, except that:

- The `–l` *library* operands have the format of options, but their position within a list of operands affects the order in which libraries are searched.
- The order of specifying the multiple `–L` options is significant.
- Conforming applications shall specify each option separately; that is, grouping option letters (e.g., `–cg`) need not be recognized by all implementations.

The following options shall be supported by the implementation:

`-c` Suppress the link-edit phase of the compilation, and do not remove any object files that are produced.

`-g` Produce symbolic information in the object or executable files; the nature of this information is unspecified, and may be modified by implementation-defined interactions with other options.

`-s` Produce object and/or executable files from which symbolic and other information not required for proper execution using the POSIX.1 {8} *exec* family has been removed (stripped). If both `-g` and `-s` options are present, the action taken is unspecified.

`-o` *outfile* Use the pathname *outfile*, instead of the default `a.out`, for the executable file produced. If the `-o` option is present with `-c`, the result is unspecified.

`-L` *directory*
Change the algorithm of searching for the libraries named in `-l` operands to look in the directory named by the *directory* pathname before looking in the usual places. Directories named in `-L` options shall be searched in the specified order. Implementations shall support at least ten instances of this option in a single `fort77` command invocation. If a directory specified by a `-L` option contains a file named `libf.a`, the results are unspecified.

`-O` *optlevel* Specify the level of code optimization. If the *optlevel* option-argument is the digit `0`, all special code optimizations shall be disabled. If it is the digit `1`, the nature of the optimization is unspecified. If the `-O` option is omitted, the nature of the system's default optimization is unspecified. It is unspecified whether code generated in the presence of the `-O 0` option is the same as that generated when `-O` is omitted. Other *optlevel* values may be supported.

`-w` Suppress warnings.

Multiple instances of `-L` options can be specified.

C.2.4 Operands

An *operand* is either in the form of a pathname or the form `-l` *library*. At least one operand of the pathname form shall be specified. The following operands shall be supported by the implementation:

file.`f` The pathname of a FORTRAN source file to be compiled and optionally passed to the link editor. The file name operand shall be of this form if the `-c` option is used.

file.`a` A library of object files typically produced by `ar` (see 6.1), and passed directly to the link editor. Implementations may recognize implementation-defined suffixes other than `.a` as denoting object file libraries.

file.o — An object file produced by fort77 −c, and passed directly to the link editor. Implementations may recognize implementation-defined suffixes other than .o as denoting object files.

The processing of other files is implementation defined.

−l *library* — (The letter ell.) Search the library named:

lib*library* .a

A library is searched when its name is encountered, so the placement of a −l operand is significant. Several standard libraries can be specified in this manner, as described in C.2.7. Implementations may recognize implementation-defined suffixes other than .a as denoting libraries.

C.2.5 External Influences

C.2.5.1 Standard Input

None.

C.2.5.2 Input Files

The input file shall be one of the following: a text file containing FORTRAN source code; an object file in the format produced by fort77 −c; or a library of object files, in the format produced by archiving zero or more object files, using ar. Implementations may supply additional utilities that produce files in these formats. Additional input files are implementation defined.

A <tab> character encountered within the first six characters on a line of source code shall cause the compiler to interpret the following character as if it were the seventh character on the line (i.e., in column 7).

C.2.5.3 Environment Variables

The following environment variables shall affect the execution of fort77:

LANG — This variable shall determine the locale to use for the locale categories when both **LC_ALL** and the corresponding environment variable (beginning with **LC_**) do not specify a locale. See 2.6.

LC_ALL — This variable shall determine the locale to be used to override any values for locale categories specified by the settings of **LANG** or any environment variables beginning with **LC_**.

LC_CTYPE This variable shall determine the locale for the interpretation of sequences of bytes of text data as characters (e.g., single- versus multibyte characters in arguments and input files).

LC_MESSAGES This variable shall determine the language in which messages should be written.

TMPDIR This variable shall be interpreted as a pathname that should override the default directory for temporary files, if any.

C.2.5.4 Asynchronous Events

Default.

C.2.6 External Effects

C.2.6.1 Standard Output

None.

C.2.6.2 Standard Error

Used only for diagnostic messages. If more than one file operand ending in `.f` (or possibly other unspecified suffixes) is given, for each such file:

`"%s:\n"`, *<file>*

may be written to allow identification of the diagnostic message with the appropriate input file.

This utility may produce warning messages about certain conditions that do not warrant returning an error (nonzero) exit value.

C.2.6.3 Output Files

Object files, listing files, and/or executable files shall be produced in unspecified formats.

C.2.7 Extended Description

C.2.7.1 Standard Libraries

The `fort77` utility shall recognize the following `−l` operand for the standard library: